Tutorial on HLA RTI 1.3

Integrated Training Program

Defense Modeling & Simulation Office
(703) 998-0660 Fax (703) 998-0667
hla@msis.dmso.mil
http://www.dmso.mil/

Current as of 01 April 98

HLA Is DoD's Technical Architecture for Modeling & Simulation

- The High Level Architecture (HLA) was approved as DoD's technical architecture for modeling and simulation on 10 September 1996
- The HLA is intended to be applied to a variety of kinds of distributed simulations, among them:
  - Analytic
  - Human-in-the-loop
  - Engineering
- The HLA contains a runtime infrastructure (RTI) component, which is the chief subject of this briefing

DMSO Has Sponsored Development of Several Versions of the RTI

- F.0 is the “familiarization version” funded by DMSO and delivered in December 1996
- F.0 implements most of the HLA Interface Spec, except:
  - Data distribution management
  - Advanced federation management, e.g. save and restore
- F.0 is designed to run on any IP network and to be ported easily among Unix boxes
- F.0 was designed primarily to be correct, but the need for speed was not ignored
- 1.0 completes implementation of Federation Management services and has improved performance
- 1.3 implements Data Distribution Management

Tutorial Is Designed to Help You Understand the RTI Generally and 1.3 in Particular

- We’ll use RTI 1.3 as an example: something concrete makes explanations easier
- We’ll try to point out what’s specific to the 1.3 design and what’s inherent in the RTI Interface Specification
- The code examples are from the Java API

Some Terminology

- **Federation:** a set of simulations, a common federation object model, and supporting RTI, that are used together to form a larger model or simulation
- **Federate:** a member of a federation; one simulation
  - Could represent one platform, like a cockpit simulator
  - Could represent an aggregate, like an entire national simulation of air traffic flow
- **Federation Execution:** a session of a federation executing together

Some More Terminology

- **Object**: An entity in the domain being simulated by a federation that
  - Is of interest to more than one federate
  - Is handled by the Runtime Infrastructure
- **Interaction**: a non-persistent, time-tagged event generated by one federate and received by others (through RTI)
- **Attribute**: A named datum (defined in Federation Object Model) associated with each instance of a class of objects
- **Parameter**: A named datum (defined in Federation Object Model) associated with each instance of a class of interactions

Rationale for HLA Design: Composability

- ***Basic premises***
  - No single, monolithic simulation can satisfy the needs of all users
  - All uses of simulations and useful ways of combining them cannot be anticipated in advance
  - Future technological capabilities and a variety of operating configurations must be accommodated
- ***Consequence*: Need composable approach to constructing simulation federations**
- ***Resulting design principles***
  - Federations of simulations constructed from modular components with well-defined functionality and interfaces
  - Specific simulation functionality separated from general purpose supporting runtime infrastructure

HLA Comprises Three Components: Rules, Runtime Infrastructure, Templates

- **HLA Rules: A set of rules which must be followed to achieve proper interaction of federates during a federation execution. These describe the responsibilities of federates and of the runtime infrastructure in HLA federations**
- **Interface Specification: Definition of the interface services between the runtime infrastructure and the federates subject to the HLA**
- **Object Model Templates: The prescribed common method for recording the information contained in the required HLA Object Model for each federation and federate**

A Federation
Must Play by the Rules

1. **Federations shall have an HLA Federation Object Model (FOM), documented in accordance with the HLA Object Model Template (OMT)**

2. **In a federation, all representation of objects in the FOM shall be in the federates, not in the runtime infrastructure (RTI)**

3. **During a federation execution, all exchange of FOM data among federates shall occur via the RTI**

4. **During a federation execution, federates shall interact with the runtime infrastructure (RTI) in accordance with the HLA interface specification**

5. **During a federation execution, an attribute of an instance of an object shall be owned by only one federate at any given time**

Each Federate Must Play By the Rules

6. Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA Object Model Template (OMT)

7. Federates shall be able to update and/or reflect any attributes of objects in their SOM and send and/or receive SOM object interactions externally, as specified in their SOM

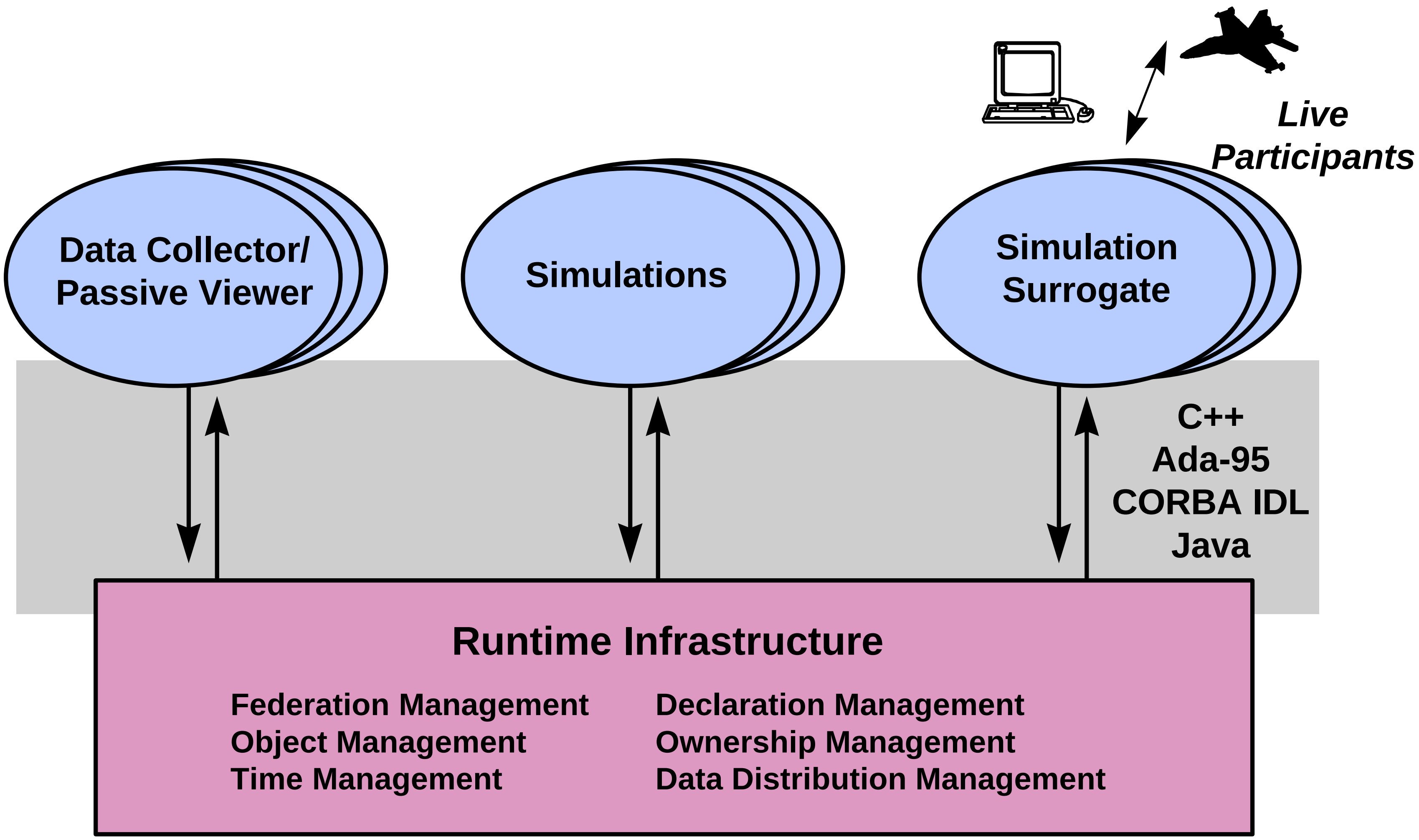
8. Federates shall be able to transfer and/or accept ownership of attributes dynamically during a federation execution, as specified in their SOM

9. Federates shall be able to vary the conditions (e.g., thresholds) under which they provide updates of attributes of objects, as specified in their SOM

10. Federates shall be able to manage local time in a way which will allow them to coordinate data exchange with other members of a federation

Architecture Splits Functions Between Simulations and Runtime Infrastructure

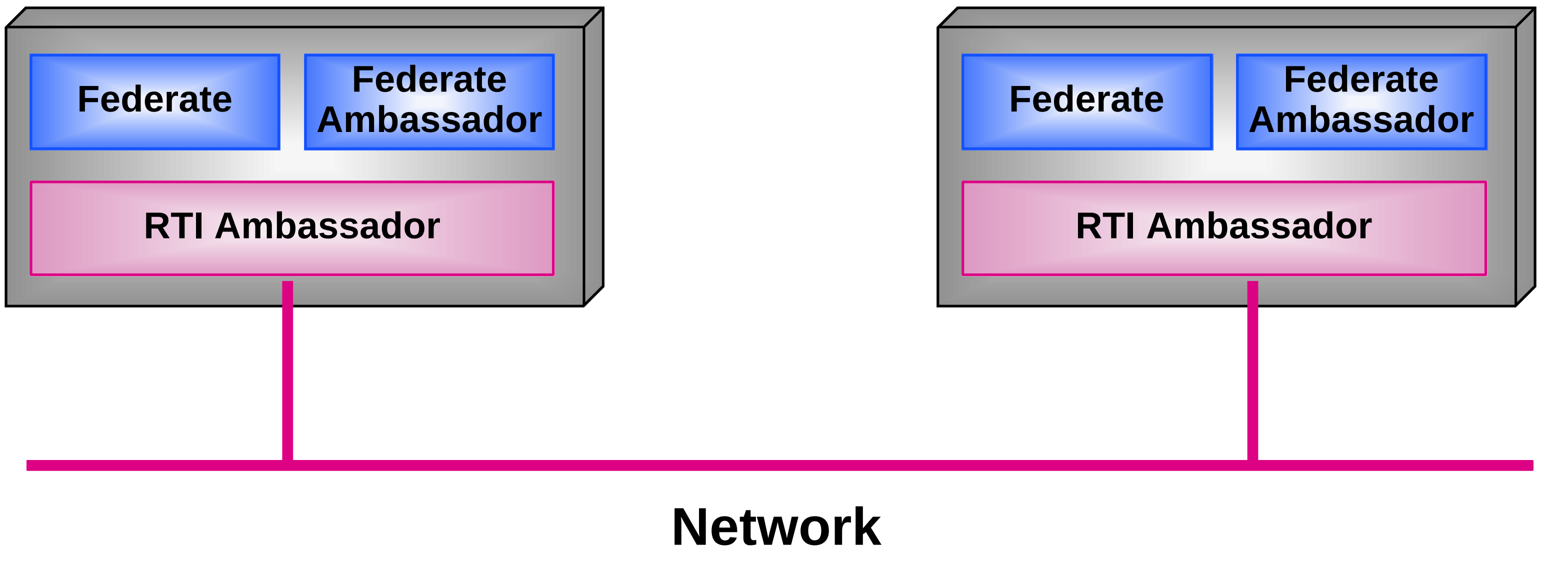
Integrated Training Program



Run-Time Infrastructure Provides Six Categories of Services

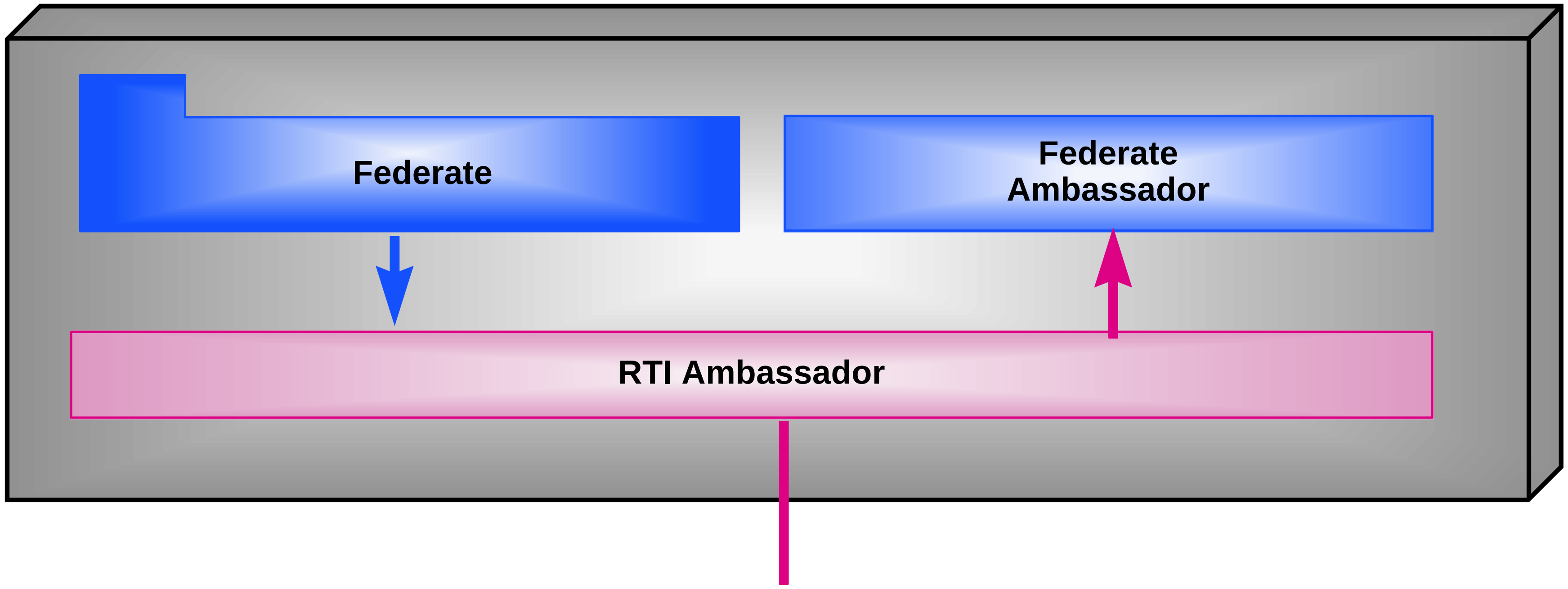
Category	Functionality
Federation Management	• Create and delete federation executions • Join and resign federation executions • Control checkpoint, pause, resume, restart
Declaration Management	• Establish intent to publish and subscribe to object attributes and interactions
Object Management	• Create and delete object instances • Control attribute and interaction publication • Create and delete object reflections
Ownership Management	• Transfer ownership of objects/attributes
Time Management	• Coordinate the advance of logical time and its relationship to real time
Data Distribution Management	• Controls the efficient routing of information between federates

In RTI 1.3, a Federate Process Contains Federate and RTI Code



Some RTI Services Are Initiated by Federate, Some by RTI

- Interfaces are method calls on objects (C++, Ada-95, IDL, Java)
- Federate-initiated serves are invoked on an instance of RTIambassador
- RTI-initiated services are invoked on an instance of FederateAmbassador



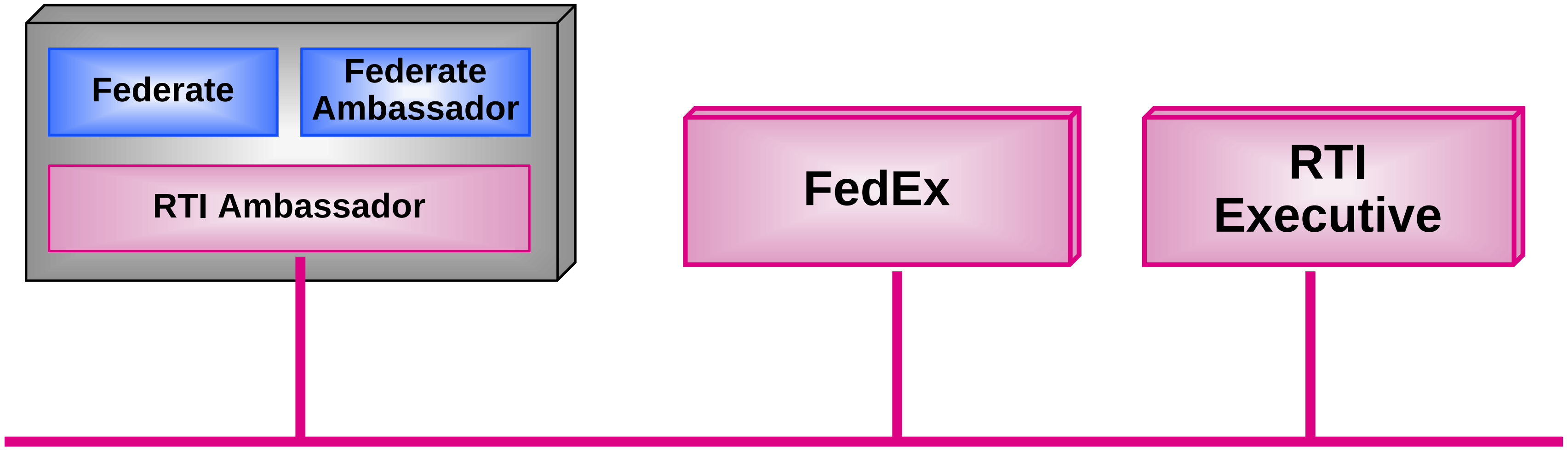
In the Beginning, There's an RTI Executive...

RTI Executive

- RTI Executive is a 1.3-supplied server started by hand
- It's an artifact of 1.3 design, not inherent in RTIs
- There is usually 1 Executive per LAN
- RTI Executive is a naming service for federation executions

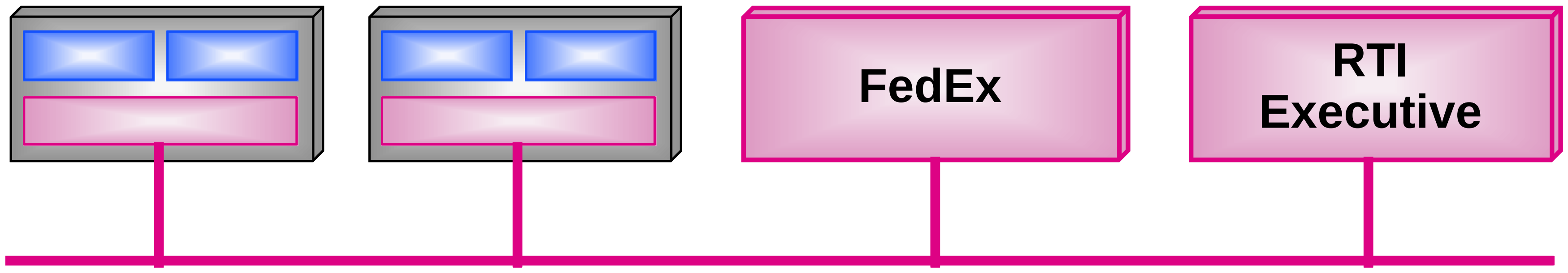
A Federate, Acting as a Federation Execution Manager, Creates a Federation Execution

- Federate invokes createFederationExecution on its RTI Ambassador
- RTI Ambassador reserves name with RTI Executive
- RTI Ambassador spawns FedEx process
- FedEx registers its communication address with RTI Executive



Other Federates Join the Federation Execution

- Another federate invokes joinFederationExecution on its RTI Ambassador
- RTI Ambassador consults RTI Executive for address of FedEx
- RTI Ambassador invokes joinFederationExecution on FedEx
- This arrangement of separate processes for RTIexec and FedEx is an RTI 1.3 artifact

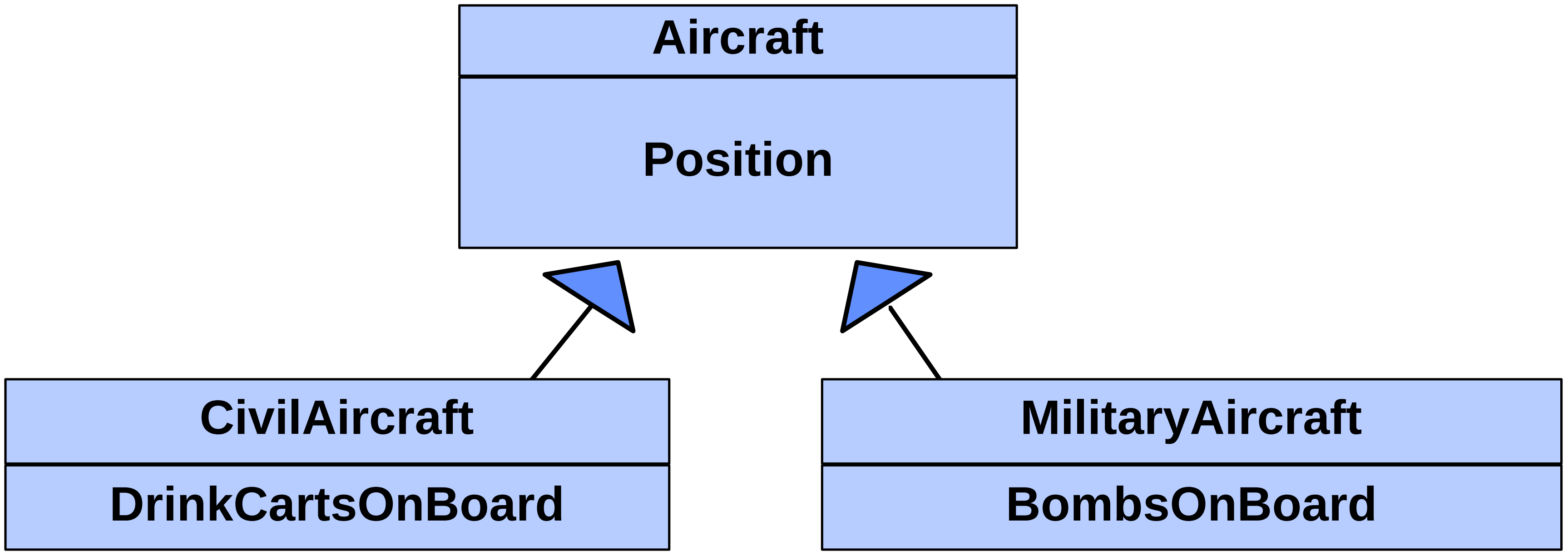


Federation Object Model (FOM) Defines Each Federation's Realm of Discourse

- Data passed between federates by the RTI is entirely parametric to the RTI
- This contrasts with DIS, where a new data item means a new PDU and changes to DIS infrastructure
- FOM describes the kinds of things federates will talk about in a federation including: objects and interactions
- FOM is agreed by federation designers before execution
- Parts of the FOM are supplied, at execution time, as data to the RTI
  - In 1.3, the FOM data takes the form of <fedexname>.fed, for "federation execution data"
  - File must be stored in a place accessible to each federate; where federate looks is a configuration item

FOM Defines Classes of Objects

- Each object class specifies the set of attributes for each instance of that class
- One object class can inherit from another



A Modest Example of Federation Execution Data (FED)

- **The FED contains the RTI-relevant part of the FOM**

```
(FED ...
  (objects
    (objectRoot …
      (attribute privilegeToDelete  reliable    timestamp)
      (class Aircraft
        (attribute Position         best_effort timestamp))
        (class CivilAircraft
          (attribute DrinkCartsOnBoard ... ))
        (class MilitaryAircraft
          (attribute BombsOnBoard ... ))
...
)
```

Interactions
Represent Events in Time

- An interaction represents an event in time that has no continuing state, e.g. a change in an ATC clearance or the firing of a weapon
- An interaction, unlike an object does not persist; it occurs at a specified time
- Federates subscribe to classes of interactions. They are then notified when another federate sends an interaction of that class
- Each interaction class specifies the set of parameters for each instance of that class
- Like object classes, one interaction class can inherit from another

Federation Management Services

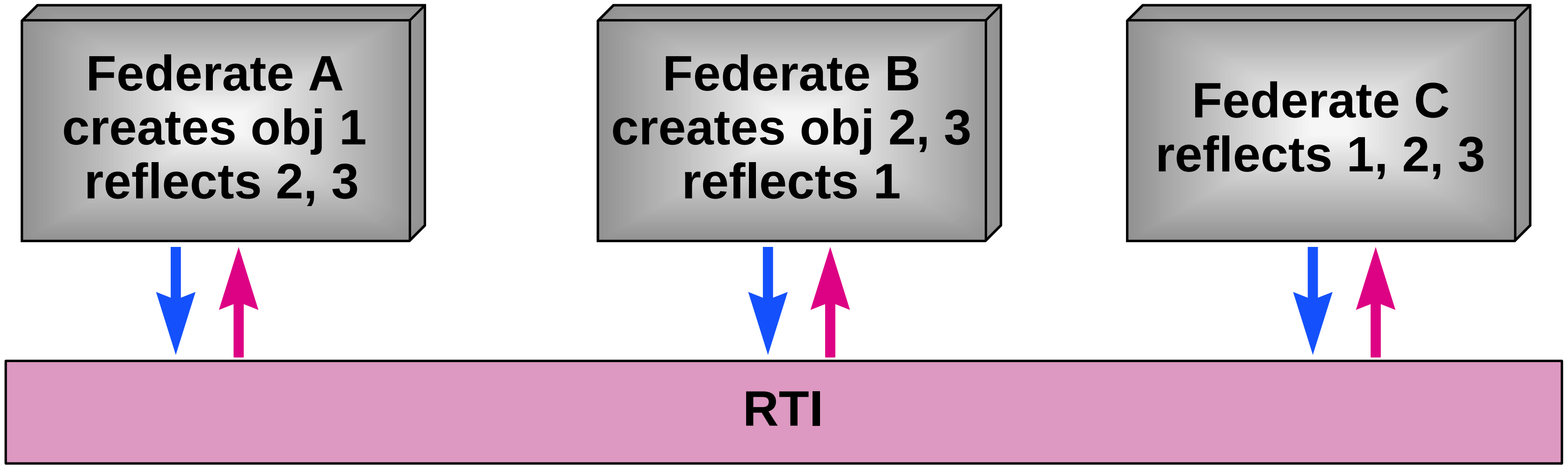
4.2 Create Federation Execution
4.3 Destroy Federation Execution
4.4 Join Federation Execution
4.5 Resign Federation Execution
4.6 Register Federation Synchronization Point
4.7 Confirm Synchronization Point Registration †
4.8 Announce Synchronization Point †
4.9 Synchronization Point Achieved
4.10 Federation Synchronized †
4.11 Request Federation Save
4.12 Initiate Federate Save †
4.13 Federate Save Begun
4.14 Federate Save Complete
4.15 Federation Saved †
4.16 Request Federation Restore
4.17 Confirm Federation Restoration Request †
4.18 Federation Restore Begun †
4.19 Initiate Federate Restore †
4.20 Federate Restore Complete
4.21 Federation Restored †

Object Management: Creating and Updating Objects, Sending and Receiving Interactions

- **Refers to management of objects in the RTI: instances of classes defined in the FOM**
- **Federates create and destroy instances dynamically. Instances have IDs unique across the life of a federation execution**
  - **int objectHandle = rtiamb.registerObject(airClassHandle);**
- **RTI signals federate when relevant instances are created by other federates: creates a reflection**

```
public void discoverObject (
          int     objectHandle,
          int     objectClassHandle)
 throws
   CouldNotDiscover,
   ObjectClassNotKnown,
   FederateInternalError;
```

Object Management Lets Federates See Each Other's Objects



- This is a publish-and-subscribe service
  - It's parameterized by FED at execution time,
  - Adjusted by each federate dynamically,
  - It's subject to time management (about which more later).

Object Management Services

5.2 Publish Object Class

5.3 Unpublish Object Class

5.4 Publish Interaction Class

5.5 Unpublish Interaction Class

5.6 Subscribe Object Class Attributes

5.7 Unsubscribe Object Class

5.8 Subscribe Interaction Class

5.9 Unsubscribe Interaction Class

5.10 Start Registration For Object Class †

5.11 Stop Registration For Object Class †

5.12 Turn Interactions On †

5.13 Turn Interactions Off †

A Contract for Data Generation and Reception is Made Using *Declaration Management*

- To begin receiving updates of an attribute, a federate must declare its interest in the attribute (It can also declare the end of its interest)
- Applies as well to classes of interactions
- The RTI uses that information to tell producing federates whether to bother updating an attribute or producing interactions of a given class

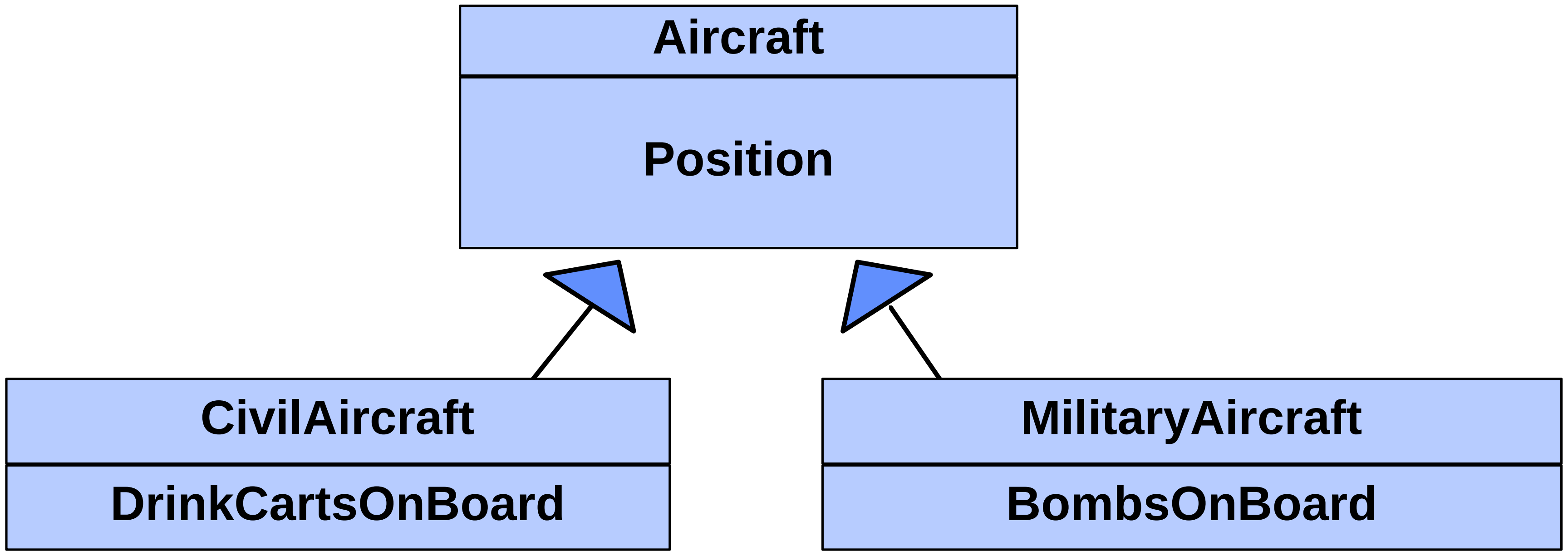
Example: Publish and Subscribe a Class

- “Publish” says the federate intends to create instances of the class and update attributes of those instances
- “Subscribe” says the federate desires to reflect some attributes of the class
- In the example below, the federate publishes and subscribes the same attribute of the same class: it’s creating some aircraft and reflecting others

```
int airClassHandle = rtiamb.getObjectClassHandle("Aircraft");
int airPVAhandle = rtiamb.getAttributeHandle("Position",
  airClassHandle);
AttributeHandleSet airAttrSet = AHsetFactory.create();
airAttrSet.add(airPVAhandle);
rtiamb.subscribeObjectClassAttribute(
  airClassHandle, airAttrSet, true /* active subscription */);
rtiamb.publishObjectClass(airClassHandle, airAttrSet);
```

Inheritance in FOM Classes Insulates Federates From FOM Changes

- **If a federate subscribes to attributes of “Aircraft” it will reflect those attributes of any subclasses**
- **Therefore FOMs can be extended (subclassed) without unconcerned federates having to change**



Exchanging Data with Other Federates Means Updating and Reflecting Attributes

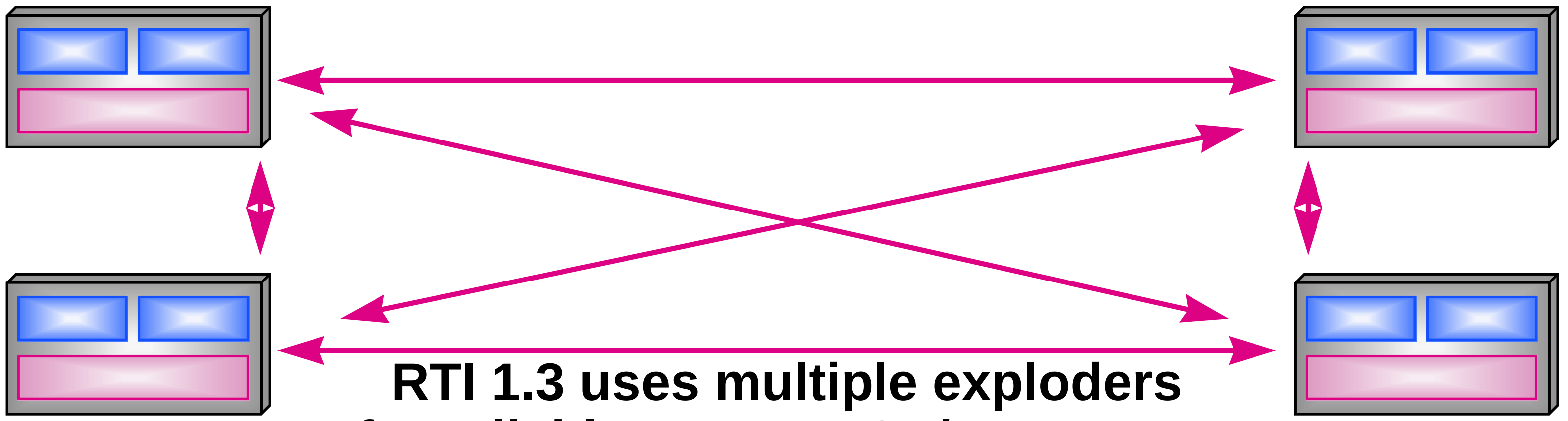
- **Update an attribute of a registered object:**

```
 ahvp.empty();
 ahvp.add(airPVAhandle, someData, 0, someData.length);
 int eventRetractionHandle = rtiamb.updateAttributeValues(
      objectId, ahvp, lastGrantedTime + increment, userTag);
```

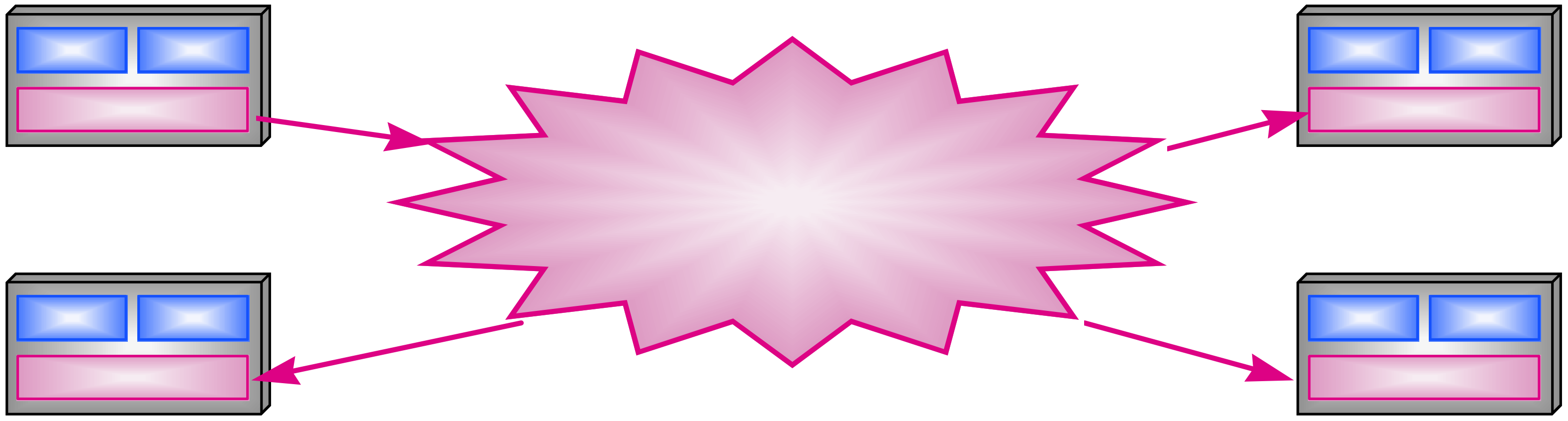
- **Receive (“reflect”) an update on a federate ambassador:**

```
public void reflectAttributeValues (
  int                           objectHandle,
  AttributeHandleValuePairSet   theAttributes,
  FederationTime                theTime,
  UserSuppliedTag               theTag,
  int                           eventRetractionHandle)
throws
  ObjectNotKnown,
  AttributeNotKnown,
  InvalidFederationTime,
  FederateInternalError;
```

Transport for Attribute Updates and Interactions is Either Reliable or Best-Effort



RTI 1.3 uses multiple exploders
for reliable comm: TCP/IP streams

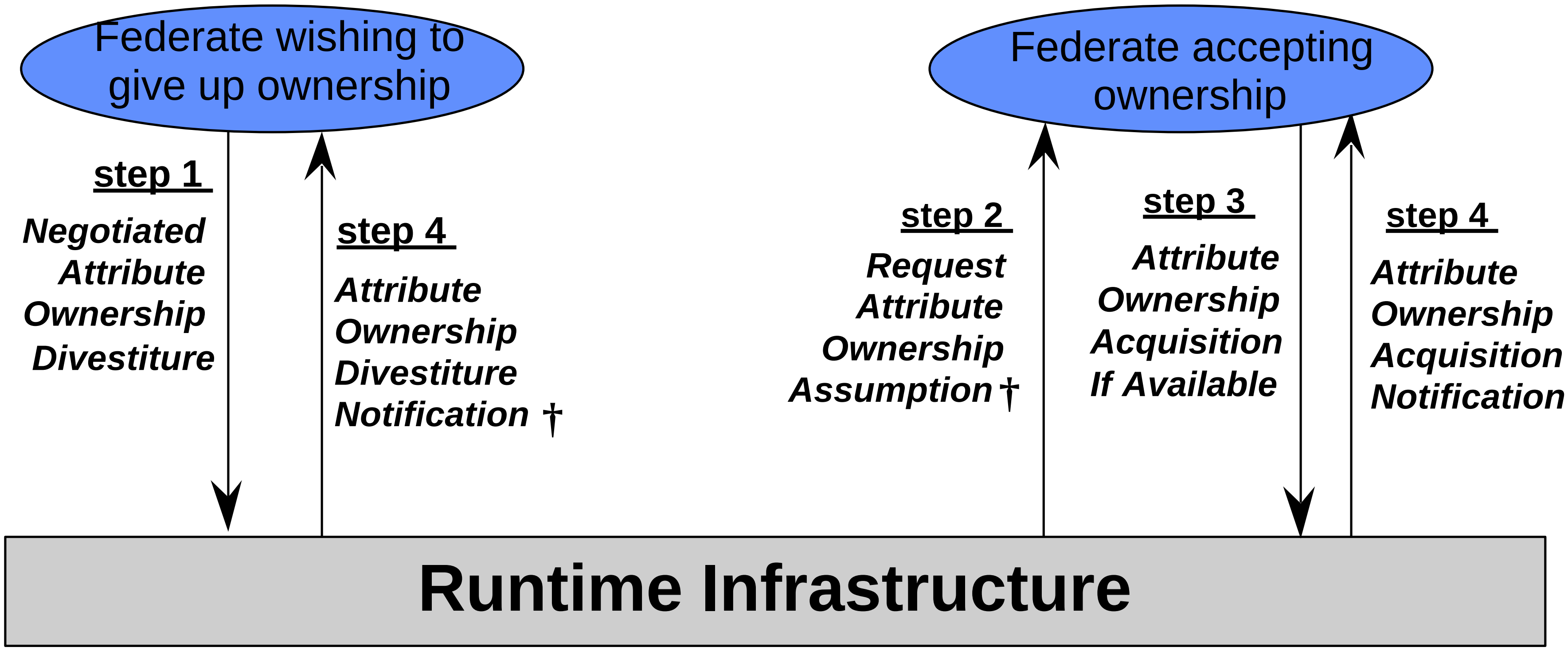


Best-effort traffic via IP multicast

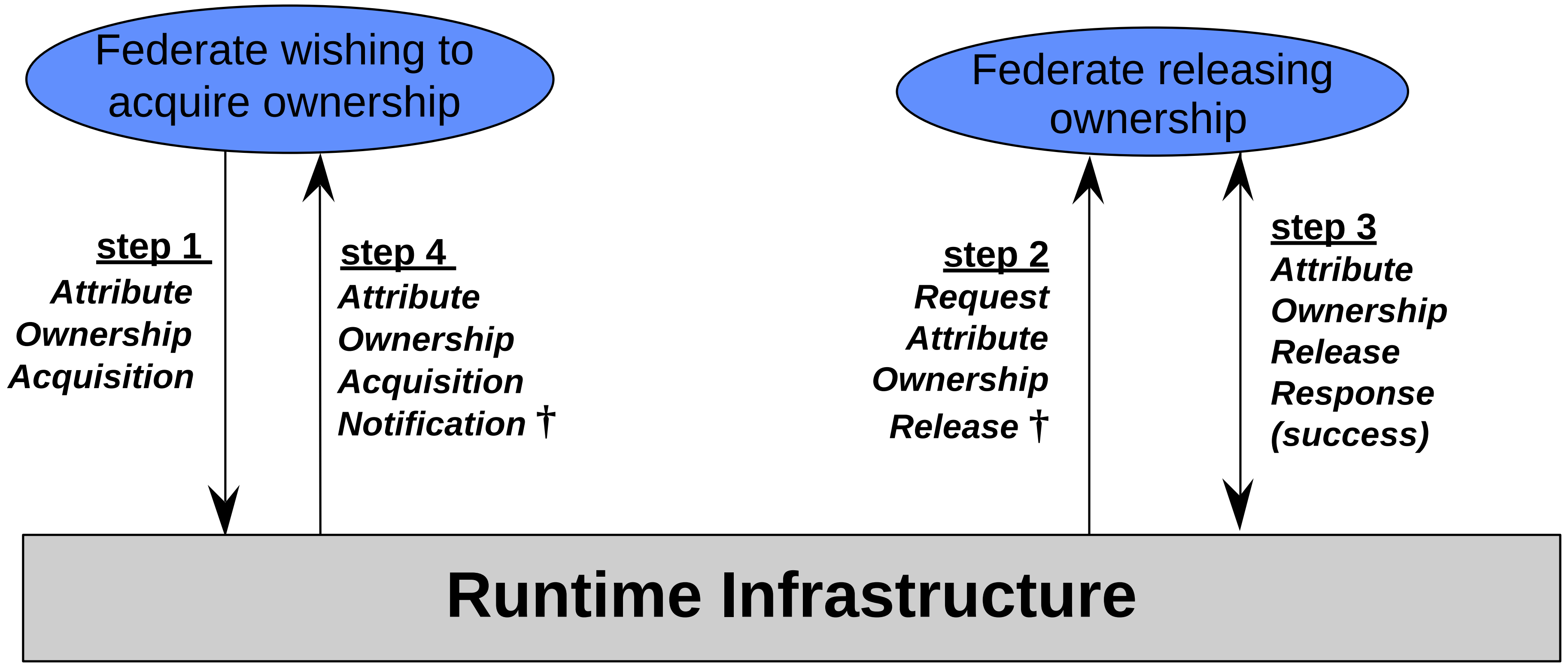
Ownership Management Allows Shared Responsibility for Simulating an Object

- Each attribute of each object has an owner: the owner is the federate responsible for updating that attribute
- Different attributes of the same object may have different owners. One federate is updating an aircraft’s position; another federate, subscribing to the position, is updating icing
- “Privilege to delete” an object is an attribute owned by some federate
- Ownership can change: responsibility for updating an attribute can pass from one federate to another. E.g., an aircraft modeled in an aggregate simulation could transfer ownership of the position attribute to a cockpit
- Ownership exchange may be pushed or pulled

Ownership May Be Pushed to Another Federate


Federate wishing to give up ownership
Federate accepting ownership
step 1
Negotiated Attribute Ownership Divestiture
step 4
Attribute Ownership Divestiture Notification †
step 2
Request Attribute Ownership Assumption †
step 3
Attribute Ownership Acquisition If Available
step 4
Attribute Ownership Acquisition Notification
Runtime Infrastructure

Ownership May Be Pulled from Another Federate


Federate wishing to acquire ownership
Federate releasing ownership
step 1
Attribute Ownership Acquisition
step 4
Attribute Ownership Acquisition Notification †
step 2
Request Attribute Ownership Release †
step 3
Attribute Ownership Release Response (success)
Runtime Infrastructure

Time Management Seeks to Synchronize the Logical Time of Federates

- Events have associated times:
  - Update of attributes
  - Sending of interactions
  - Deletion of objects
- Fundamental problem is to ensure that either:
  - No federate receives an event in its past (conservative approach) or
  - A federate that has computed into the future, when it receives an event in its past that invalidates its state, has the information necessary to roll back to the time of the event (optimistic approach)

Time Management

- Another approach, used in “real-time” simulations *à la* DIS, is to ignore time management and deem events to occur when they are perceived
  - Federate’s time is computed from its host wall clock
  - This cannot guarantee strict causality
  - If latencies on the delivery of events are bounded, this may be good enough for human perception
- The Interface Specification characterizes time management by two Boolean “switches” set by each federate:
  - Time-regulating
  - Time-constrained

Time Management Seeks to Accommodate Variety of Schemes in a Single Federation

		Time-Regulating	
		true	false
Time-Constrained	true	Strictly Time-Synchronized: conservative (ALSP) and aggressive (Time Warp)	Viewer or Federation Management Tool: stays synchronized to federation, but generates no events
	false	Unconstrained (DIS) operating with conservative federates	Externally Synchronized Simulation: no time management from RTI’s standpoint (DIS)

Optimistic Federates Must Be Ready to Retract Their Events

- An optimistic federate turns time-regulating and time-constrained on. The optimistic federate invokes FlushQueueRequest, is given all pending events, and is granted to latest safe time
- Each time the federate creates an event (creates or destroys an object, updates an attribute, sends an interaction), it receives from its RTI ambassador an EventRetractionHandle
  - Federate must remember the retraction handles
  - If an event arrives in the federate’s past causing it to roll back, it must invoke rtiamb.retract(EventRetractionHandle)
  - Any RTI Ambassador that has delivered the event to its federate will invoke reflectRetraction(ERH) on its Federate Ambassador

Receive-Order Attributes and Interactions Bypass the RTI Ambassador Time Queues

- Attributes and interactions can be designated as receive-order rather than time-stamp-order
- Receive-order attributes and interactions are delivered immediately by the RTI Ambassador to the federate, irrespective of the federate’s value of local time
- A federate operating with “time regulating” off will send all its attribute updates and interactions in receive order, irrespective of the FED specification
- A federate operating with “time constrained” off will treat all arriving attribute updates and interactions as receive-order, irrespective of the sending federate’s specification

Management Object Model (MOM) Exploits RTI for Management of Federation Execution

- Federation Executions are managed by a combination of Federate- and RTI-supplied information
- This information can be structured using the same tools used for simulation data
- The MOM defines classes and interactions related to federation management just as the FOM defines classes and interactions in the simulation domain
- A manager federate can
  - Monitor and control aspects of the federation through the MOM
  - Subscribe to MOM object classes and interactions exactly as it would to parts of any FOM
- The RTI-supplied aspects of the MOM will be standardized
- Federate-supplied MOM data depends on the federation needs

MOM Data Must Be Included in Any FED File

```
(FED
 (...
   )
   (class Manager
     (class Federate
       ...
       (attribute FederateHandle reliable receive)
       (attribute FederateType   reliable receive)
       (attribute FederateHost   reliable receive)
       (attribute RTIversion     reliable receive)
       (attribute FEDid          reliable receive)
       (attribute TimeContrained reliable receive)
       (attribute TimeRegulating reliable receive)
       ...
     )
   )
 )
...
)
```

An object of this class is created each time a federate joins

Manager federate can subscribe to these attributes

Interactions to Control the Federation Execution

```
(FED
 (objects
   ...
 )
 (interactions
   (class Manager                    reliable receive
     (class Federate                 reliable receive
       (parameter Federate)
       ...
       (class Adjust                 reliable receive
         ...
         (class ModifyAttributeState reliable receive
           (parameter ObjectInstance)
           (parameter Attribute)
           (parameter AttributeState)
           ...
         )
       )
     )
   )
 )
)
```

By sending interactions with values for these parameters, manager federates can change behavior of another portion of the RTI

“Process Model” Refers to the Way Threads of Control Interact Between Federate and RTI

Integrated Training Program

- The process model of an RTI is not defined by the Interface Specification
- A design requirement of 1.3 is that it support a strictly single-threaded federate
- RTI Ambassador is in the federate’s process space; in a single-threaded language, it gets no thread of control unless federate gives it one
- Federate calls rtiamb.tick() periodically to allow RTI Ambassador to read its sockets and to initiate callbacks on the Federate Ambassador
- Federate is prevented from calling the RTI Ambassador recursively from a callback (in most instances)

Updating the Attributes of an Object

1 Is this object known to the RTI?

2 Has the class been published?

3 Are all the attributes owned?

4 Is the time valid?

5 Create a retraction handle and return the service call.

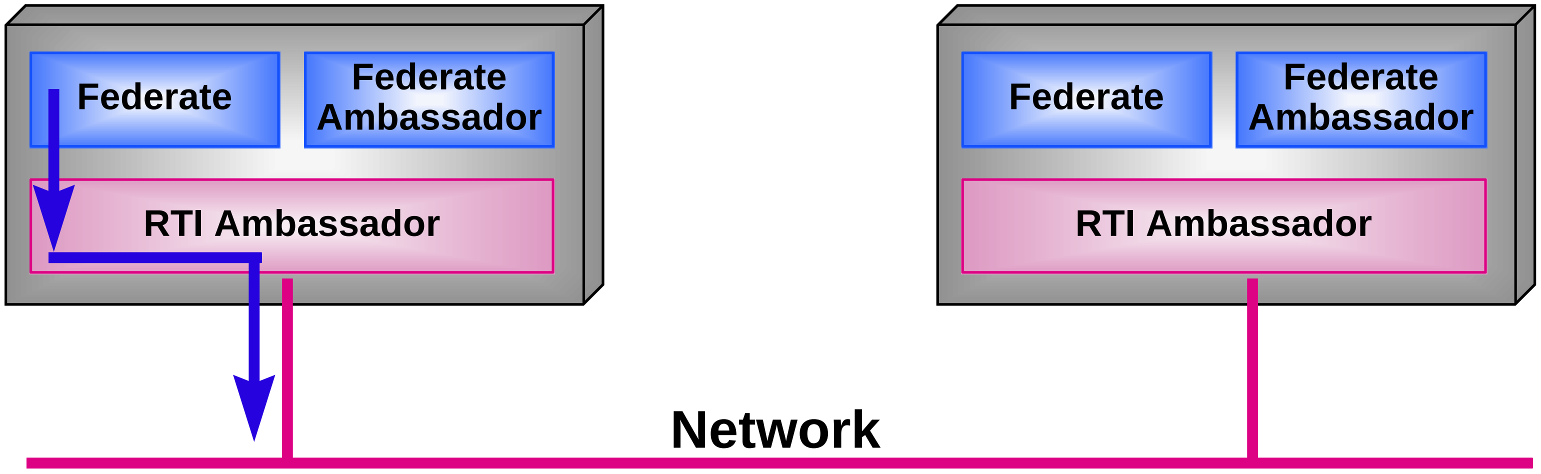
6 Assign desired transportation category to each attribute.

7 Find ordering category and assign to each attribute.

8 If regulation off, then force ordering category to arrival order.

9 Write best-effort attributes to one of the IP multicast group assigned for the federation execution.

10 Send reliable attributes to reliable distributor.



Receiving
Attributes of an Object

1 Accept a new message from either an IP multicast group or a reliable distributor.

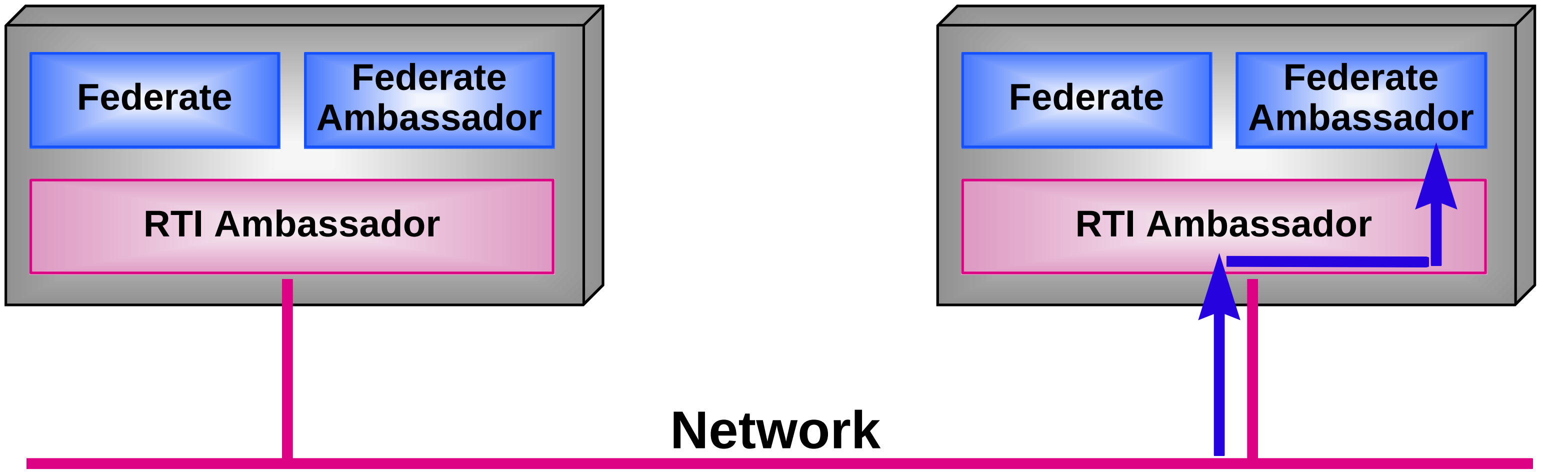
2 If constrained ON and update is TSO then place in the TSO queue; else place in the FIFO queue.

3 At right time, remove appropriate messages from the queues.

4 If registered object class is not a subclass subscribed to by the federate, then discard the update.

5 Promote the registered class to a represented class and remove any inappropriate attributes.

6 Provide attributes to the federate.



For Further Information...

- Main HLA Web page: http://www.dmso.mil/projects/hla/
  - HLA Overview Briefing (Annotated)
  - AMG Briefings
  - Related SIW publications
  - Dr. Kaminski’s HLA mandate memo
  - RTI software request form
- Send comments or questions to: hla @dmso.mil